SE499 Report — Path Following Controllers

Rollen S. D'Souza
Software Engineering
Department of Electrical & Computer Engineering
`rs2dsouz@edu.uwaterloo.ca`

Acknowledgements

The author thanks Professor Christopher Nielsen for his guidance in developing the required intuition and mathematical tools for path following control design. Plots and simulations were partly written in Mathworks MATLAB under a student licence. Other simulations were written using C++ linked with the Boost library [2] and Catch test framework [8].

Rollen S. D'Souza
Software Engineering
University of Waterloo
rs2dsouz@edu.uwaterloo.ca

Contents

Nomenclature

$B_\epsilon(\boldsymbol{x})$	Open ball of radius ϵ centred about $\boldsymbol{x}$ in the appropriate vector space.
$\boldsymbol{R}_\theta$	Counter-clockwise rotation matrix in $\mathbb{R}^2$ of angle θ.
$\mathcal{T}$	Mobile robot task space.
$\mathcal{T}_{(w,h)}$	Rectangular mobile robot task space with width w and height h.
$\mathbf{M}$	A matrix. Can be assumed real unless stated otherwise.
$\mathbf{M}_{(i,j)}$	The value at the i-th row and j-th column of matrix $\mathbf{M}$. The variables may have a : substituted to indicate selection of all values in that dimension, similar to that found in the Matlab grammar.
$\boldsymbol{x}$	A vector in $\mathbb{R}^n$.
$\boldsymbol{x}_i$	The i-th component of vector $\boldsymbol{x}$

List of Figures

List of Tables

Chapter 1

Introduction

1.1   Background

A common control objective for mobile robots involves tracking a trajectory in the space the robot operates in. This space is known as the task space of the robot, denoted as $\mathcal{T}$. Often the structure of this space is unknown — along with the goal — and the robot must first explore the world using an exploration and mapping algorithm before deciding on a goal and path. This paper is concerned with a restricted subset of this problem. Consider a differential drive robot starting at location $(0,0) \in \mathcal{T}$, placed in a world with an unknown number of obstacles, that is given the objective to reach position (x, y) in task space without colliding with obstacles. A two-pronged approach can be taken to safe-guard from collisions:

1. Plan a path that can be followed successfully by the robot and does not intersect with any obstacles for all future time.

2. Design a controller that ensures sufficiently fast convergence to the path and provides a guarantee, under reasonable assumptions, that the robot will not leave the path.

Planning a path is a field in its own right. The most practical, and dominant, philosophy to path planning involves sampling $\mathcal{T}$ and incrementally building a path to the goal region. The sampling algorithm used is critical to performance and effective coverage of $\mathcal{T}$. If the sampling technique does not adequately sample the space in question the algorithm may take considerably longer to generate a path that reaches $\mathcal{G}$. Incorporated in these algorithms is the ability to test the feasibility of a path. For example, whether any collisions would occur with the known environment and whether any differential constraints of the robot are broken. For example, the robot cannot turn instantaneously. This addition ensures that unfeasible paths are rejected early instead of wasting time developing a path that is known to be unfeasible. Of course, the astute reader notes that this type of path generation often creates discrete segments that lead to the goal. There are techniques in literature to develop smooth paths incrementally in coordination with a sampling algorithm but these are considered outside the scope of this work [1]. Other advanced approaches to generate smooth paths rely on the construction of a vector field over $\mathcal{T}$ [7] but these are also relatively complicated algorithms.

Instead this report takes a simpler approach that generates a smooth function out of the discrete waypoints produced by the relatively simple Rapidly exploring Randomized Tree (RRT) algorithm. The smooth function generated is a polynomial spline, a stitched sequence of possibly distinct polynomials that preserves a set of continuity conditions at the stitch-points. After generation, the spline is tested again for feasibility before being propagated to the controller. There are apparent performance issues in this approach addressed in Section 3.

Independently the field of non-linear control theory developed techniques to follow arbitrary paths. Standard control techniques tend to yield poor results in tracking a path. A number of techniques in literature exist to drive a robot towards a path and converge upon it. However, most of these techniques do not provide invariance once on the path [3]. This lack of invariance reduces the robustness of the technique as the controller may not stay on the path even under perfect environmental conditions! This motivated the development of a new

family of path following controllers that treat the path as an invariant set that is then stabilized through linear control, described in Section 2. This family of controllers use transverse feedback linearization to reformulate the problem and assist in constructing a linear controller that compensates for the non-linearities imposed by the path. This report discusses a few methods to planning and control for the purpose of driving a differential drive robot towards a goal.

1.2 Notation and Report Organization

This report assumes proficiency in multi-variable calculus and basic control theory. Any results taken from theory of dynamical systems and other advanced undergraduate mathematical courses are stated and cited from a source used by the author. Scalars are denoted as x, vectors as $\boldsymbol{x}$ and matrices as $\mathbf{X}$. Time derivatives are denoted using Newton's dot notation and any other derivatives are denoted using primes if the independent variable is clear and Leibniz notation otherwise. In practice, planning algorithms generally precede control. However this report instead defers discussion of planning until after describing control methodologies in Section 2. This content placement is intended to give the reader a better intuition for the decisions made at planning stage.

1.3 Problem Statement

Let the task space, without loss of generality, be the rectangular space $\mathcal{T}_{(\bar{x},\bar{y})} = \{(x,y) \in \mathbb{R}^2 : 0 \le x \le \bar{x} \wedge 0 \le y \le \bar{y}\}$. For simplicity, consider the kinematic model of a differential drive robot with the combined position and orientation state vector $\boldsymbol{x} = (x,y,\theta)^\mathsf{T} \in \mathbb{R}^3$. The robot has a controllable forward velocity $v \in \mathbb{R}$ and a controllable turning rate $u \in \mathbb{R}$. v is assumed to be positive. The kinematic system dynamics are
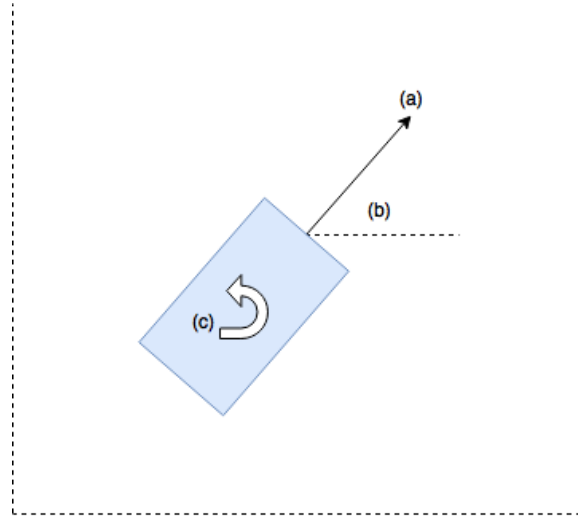
Figure 1.1: Kinematic model of differential drive robot. (a) is the direction vector along which the robot is currently facing, denoted by $\boldsymbol{g}_v$. That is a function of x_3, the angle of the vector with respect to x-axis which is shown at (b). (c) is the rate at which the robot turns and is controlled by u.

$$\mathbf{G}(\boldsymbol{x}) = \begin{bmatrix} 0 & \cos(\boldsymbol{x}_3) \\ 0 & \sin(\boldsymbol{x}_3) \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{g}_u & \boldsymbol{g}_v \end{bmatrix}$$

$$\dot{\boldsymbol{x}} = \mathbf{G}(\boldsymbol{x}) \begin{bmatrix} u \\ v \end{bmatrix}. \tag{1.1}$$

The robot must at some unspecified finite time arrive at a given goal region $\mathcal{G} \subset \mathcal{T}$. Note there is no restriction in how it arrives in this region other than it does so safely. Starting at time $t = 0$ and then at a regular time interval Δ, the robot observes the world and decides, based on the observation, whether a new plan is required to approach the goal region. This may occur when the robot observes a new obstacle that the robot's currently planned path intersects. The simplest and most widely used planning algorithms involve the generation of a tree that explores the current model of the world. This family of algorithms generate a sequence of waypoints. From planning stage a path $\boldsymbol{\sigma} : [0, 1] \subset \mathbb{R} \to \mathcal{T}$ with $\boldsymbol{\sigma}(1) \in \mathcal{G}$ is generated.

The path $\boldsymbol{\sigma}_a$ is fed into the path following control which guides the robot over time towards, and along, the path in order to arrive in $\mathcal{G}$. Further, the control algorithm must converge towards the path in a manner that is both guaranteed and as fast as possible. It must also faithfully follow the path in order to reduce the chances of re-planning. Given that the robot may not have a perfect model of the world, i.e, is not aware of all obstacles, a re-plan may still occur. The planning algorithm is required to design a new $\boldsymbol{\sigma}_b$ so as to preserve any properties the control algorithm requires to stay on the trajectory continuously.

Chapter 2

Control

Three control strategies are considered for path following. The simplest controller, in terms of design and implementation, is the point-chasing controller. The other two techniques rely on the approach of transverse feedback linearization which decomposes the path following problem into transverse and tangential dynamical systems. This section assumes the path is of the form in (2.1). The matrix representation of a quintic polynomial is chosen for succinct representation.

$$\mathbf{P} \in \mathbb{R}^{2 \times 6}$$

$$\boldsymbol{\sigma}(\lambda) = \mathbf{P} \begin{bmatrix} 1 \\ \lambda^1 \\ \lambda^2 \\ \lambda^3 \\ \lambda^4 \\ \lambda^5 \end{bmatrix}. \tag{2.1}$$

2.1 Trajectory Controller
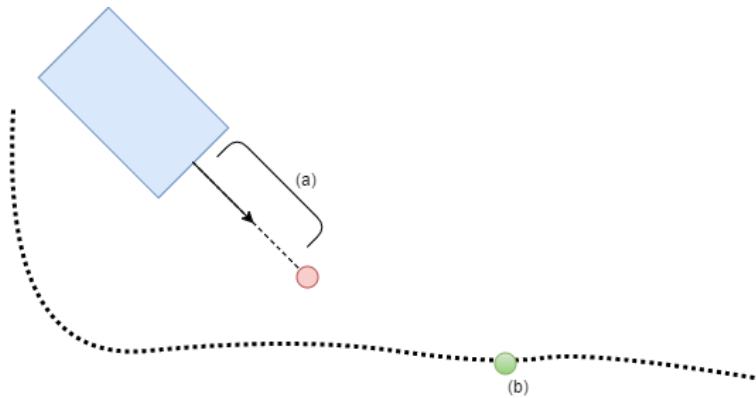
Figure 2.1: Visual description for trajectory controller. (a) is the scaled direction vector $l\boldsymbol{\tau}$ that is then added to the current position and compared to the point (b) tracing the curve.

A trajectory controller involves treating the parameter λ of the path $\boldsymbol{\sigma}$ as a function of time, t, and designing a controller that converges to this moving point. In order to apply this strategy, the control is designed around

a point that leads the robot's current position. Let the observation be defined as,

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{\tau} = \begin{bmatrix} \cos(\boldsymbol{x}_3) \\ \sin(\boldsymbol{x}_3) \end{bmatrix}$$

$$\boldsymbol{y}_l = \mathbf{H}\boldsymbol{x} + l\boldsymbol{\tau}(\boldsymbol{x}_3)$$

for some positive $l \in \mathbb{R}$. We let $\boldsymbol{y}_{ref}(t) = \boldsymbol{\sigma}\left(\lambda(t)\right)$. This is visually shown in Figure 2.1. The introduction and restriction of the parameter l is made clear in the controller derivation. Define the position error $\boldsymbol{e} := \boldsymbol{y}_l - \boldsymbol{y}_{ref}$ and take the time derivative to form a new dynamical system in terms of the path error,

$$\begin{aligned}
\dot{\boldsymbol{e}} &= \dot{\boldsymbol{y}}_l - \dot{\boldsymbol{y}}_{ref} \\
&= \mathbf{H}\left(\boldsymbol{G}(\boldsymbol{x})\begin{bmatrix} u \\ v \end{bmatrix}\right) + l\dot{\boldsymbol{\tau}} - \boldsymbol{\sigma}'(\lambda(t))\dot{\lambda}(t) \\
&= \begin{bmatrix} v\ cos\ \boldsymbol{x}_3 \\ v\ sin\ \boldsymbol{x}_3 \end{bmatrix} + \begin{bmatrix} -l\ sin\boldsymbol{x}_3 \\ l\ cos\ \boldsymbol{x}_3 \end{bmatrix}u - \boldsymbol{\sigma}'(\lambda(t))\dot{\lambda}(t) \\
&= \begin{bmatrix} -l\ \sin\boldsymbol{x}_3 & \cos\ \boldsymbol{x}_3 \\ l\ \cos\ \boldsymbol{x}_3 & \sin\ \boldsymbol{x}_3 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} - \boldsymbol{\sigma}'(\lambda(t))\dot{\lambda}(t).
\end{aligned} \tag{2.2}$$

At this point the reader may observe that the matrix multiplying the control signals is invertible if and only if l is non-zero. We now construct a controller of the form

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -l\ \sin\boldsymbol{x}_3 & \cos\ \boldsymbol{x}_3 \\ l\ \cos\ \boldsymbol{x}_3 & \sin\ \boldsymbol{x}_3 \end{bmatrix}^{-1}\left[\mathbf{K}\boldsymbol{e} + \boldsymbol{\sigma}'(\lambda(t))\dot{\lambda}(t)\right].$$

Notice that this choice of controller yields $\dot{\boldsymbol{e}} = \mathbf{K}\boldsymbol{e}$ when substituting into (2.2), forming a closed loop linear system. As long as $\mathbf{K} \in \mathbb{R}^{2\times 2}$ is Hurwitz, the closed loop system is stable. As a simplification, this paper assumes $\lambda(t) = \frac{t}{T_f}$ where $T_f > 0$ is some constant scaling factor.

2.2   Transverse Feedback Linearization

A variety of techniques were designed to address the issues observed in the point chasing controller. One of these techniques is transverse feedback linearization. Transverse feedback linearization is the mathematical reformulation of system dynamics into transverse and tangential components with respect to the closest point to the path. This restatement of the problem allows for the capturing of the path dynamics — combining that with the original system dynamics — and permits cancellation of non-linearities introduced by the path as it affects the system.

The technique was generalized in [9] for aperiodic paths and then further expanded in [4]. The approaches defined in those works restricted their design to following simple curves, such as circles. Gill et al., in [3], applied the technique to a larger class of paths, specifically splines in C^p with $p >= 2$. Note that their work is not restricted to polynomial splines and in fact applies to any functions that are stitched at waypoints preserving C^p continuity on the whole parameterized domain as well as requiring derivatives up to order p be linearly independent. As stated previously, it is assumed in this work that these splines are quintic polynomials in order to provide a relevant benchmark comparison.

2.2.1   General Form

Recall the kinematic model for a differential drive robot, from (1.1). Unlike the trajectory controller, this controller will assume that v is a fixed quantity. Therefore, the system dynamics may be rearranged into

$$\begin{aligned}
\dot{\boldsymbol{x}} &= \boldsymbol{g}_{\boldsymbol{v}}(\boldsymbol{x})v + \boldsymbol{g}_{\boldsymbol{u}}u \\
&:= \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{b}u.
\end{aligned} \tag{2.3}$$

The output of the system is defined as

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{y}(\boldsymbol{x}) = \mathbf{H}\boldsymbol{x}.$$

As before, let the path be $\boldsymbol{\sigma} : [0,1] \to \mathcal{T} \subset \mathbb{R}^2$ and C^2 everywhere. A few regularity condition are required.

Assumption 2.2.1 (Path is self-intersection free)**.**

$$\forall \lambda_1, \lambda_2 \in \mathbb{R} : \boldsymbol{\sigma}(\lambda_1) = \boldsymbol{\sigma}(\lambda_2) \Leftrightarrow \lambda_1 = \lambda_2$$

Assumption 2.2.2 (Path is a regular curve.)**.**

$$\forall \lambda \in \mathbb{R} : \boldsymbol{\sigma}'(\lambda) \neq 0$$

Take note that there is no restriction on the domain of $\boldsymbol{\sigma}$ in the assumptions although work in this report does restrict the domain. We define a new state vector $\boldsymbol{\xi} := (\xi_1, \xi_2)^\mathsf{T}$ where $\boldsymbol{\xi}_1$ is left undefined for now. One may view $\boldsymbol{\xi}_1$ as the signed tracking error of the robot with respect to the path. The variation of this state is what differentiates the variety of techniques used. When the path is well known and simple — such as a circular arc — this state may be expressed in closed form. This is not always the case. Let us then define $\boldsymbol{\xi}_2 := \dot{\boldsymbol{\xi}}_1$. Chain rule gives,

$$\begin{aligned} \boldsymbol{\xi}_2 &= \dot{\boldsymbol{\xi}}_1 \\ &= \mathrm{d}_{\boldsymbol{x}}\boldsymbol{\xi}_1 \dot{\boldsymbol{x}} \\ &= \mathrm{d}_{\boldsymbol{x}}\boldsymbol{\xi}_1 \begin{bmatrix} v\,cos(\boldsymbol{x}_3) \\ v\,sin(\boldsymbol{x}_3) \\ u \end{bmatrix}. \end{aligned}$$

Under the assumption that this error is independent of robot heading we can further simplify the expression such that $\boldsymbol{\xi}_2 = \mathrm{d}_{\boldsymbol{x}}\boldsymbol{\xi}_1 \boldsymbol{f}(\boldsymbol{x}) =: \mathcal{L}_{\boldsymbol{f}}\boldsymbol{\xi}_1(\boldsymbol{x})$. The function $\mathcal{L}_{\boldsymbol{f}}\boldsymbol{\xi}_1 : \mathbb{R}^3 \to \mathbb{R}$ is known as the Lie derivative of $\boldsymbol{\xi}_1$ along the vector field $\boldsymbol{f}(\boldsymbol{x})$. Notice the control signal has no impact on our state. The lack of a control signal in the dynamical system motivates finding the time derivative of $\boldsymbol{\xi}_2$. Using properties of Lie derivatives, it is not difficult to show that $\boldsymbol{\xi}_2 = \mathcal{L}_{\boldsymbol{b}}\mathcal{L}_{\boldsymbol{f}}\boldsymbol{\xi}_1(\boldsymbol{x})u + \mathcal{L}_{\boldsymbol{f}}^2\boldsymbol{\xi}_1(\boldsymbol{x})$. This yields the system dynamics,

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} \xi_2 \\ \mathcal{L}_{\boldsymbol{b}}\mathcal{L}_{\boldsymbol{f}}\boldsymbol{\xi}_1(\boldsymbol{x})u + \mathcal{L}_{\boldsymbol{f}}^2\boldsymbol{\xi}_1(\boldsymbol{x}) \end{bmatrix}. \tag{2.4}$$

(2.4) can be solved for the control signal. Motivated by the structure of (2.4), we take our control law to be

$$u = \frac{u_{aux} - \mathcal{L}_{\boldsymbol{f}}^2\boldsymbol{\xi}_1(\boldsymbol{x})}{\mathcal{L}_{\boldsymbol{b}}\mathcal{L}_{\boldsymbol{f}}\boldsymbol{\xi}_1(\boldsymbol{x})}, \tag{2.5}$$

where u_{aux} is an outer loop control signal that is yet to be defined. The reader may wonder under what conditions $\mathcal{L}_{\boldsymbol{b}}\mathcal{L}_{\boldsymbol{f}}\boldsymbol{\xi}_1(\boldsymbol{x})$ is non-zero. This is not apparent, although intuition may guide us. Since $\mathcal{L}_{\boldsymbol{b}}\mathcal{L}_{\boldsymbol{f}}\boldsymbol{\xi}_1(\boldsymbol{x})$ scales the control signal we expect it to vanish whenever the control signal has no instantaneous impact on the change in the orthogonal error. This may occur, for example, when the differential robot is facing towards the nearest point. Leaving the discussion of singularities aside, we set $u_{aux} := \boldsymbol{k}^\mathsf{T}\boldsymbol{\xi}$ with $\boldsymbol{k} \in \mathbb{R}^2$. Substituting u_{aux} into (2.5) and (2.4), the partial dynamical system becomes

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} 0 & 1 \\ k_1 & k_2 \end{bmatrix} \boldsymbol{\xi}. \tag{2.6}$$

This system only describes dynamics transversally as we have no control over v in the original system. (2.6) is a linear system that is stable as long as the gains $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$ are negative, as per the Routh-Hurwitz criterion.

2.2.2   Sylvester Approach

The Sylvester methodology relies on the theoretical developments of elimination theory. Elimination theory investigates the theoretical methods that discover common roots of polynomials. The essential idea is to reformulate (2.1) into a system of homogeneous linear equations that has a non-trivial kernel [10]. This non-trivial kernel allows for the derivation of an implicit representation of the path, otherwise known as a level set, that can then be stabilized by the transverse feedback linearization. First, we state a necessary result from linear algebra.

Theorem 2.2.3. *Let* $\mathbf{A} \in \mathbb{R}n \times n$ *and* $\boldsymbol{x} \in \mathbb{R}^n$. *Then the solution to* $\mathbf{A}\boldsymbol{x} = \mathbf{0}$ *is non-unique if and only if* $\det \mathbf{A} = 0$.

Notice that if we formulate a new system from (2.1) that is square and homogeneous, we will be able to eliminate $\boldsymbol{\lambda}$ by instead considering when $\det \mathbf{A} = 0$ for some constructed $\mathbf{A}$. This is otherwise known as finding the resultant [10]. Homogenizing the path requires treating $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ as constants $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$, moving them into the polynomial matrix $\mathbf{P}$. We call this modified matrix $\mathbf{Q}$. With this modification, we need only construct a square system out of $\mathbf{Q}$. In order to construct a square linear system, we consider multiplying both polynomials in the first and second component of $\boldsymbol{\sigma}$ by increasing powers of λ until we have generated a sufficient number of equations. For two quintics, we must multiply the polynomials by powers of λ up til λ^5, arriving at (2.7). The matrix S and its determinant are known as the Sylvester matrix and the Sylvester resultant respectively.

$$\begin{bmatrix} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 6} \\ 0 & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 5} \\ \mathbf{0}_{1\times 2} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 4} \\ \mathbf{0}_{1\times 3} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 3} \\ \mathbf{0}_{1\times 4} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 2} \\ \mathbf{0}_{1\times 5} & \mathbf{Q}_{1:} & \mathbf{0}_{1\times 1} \\ \mathbf{0}_{1\times 6} & \mathbf{Q}_{1:} & \\ & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 6} \\ 0 & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 5} \\ \mathbf{0}_{1\times 2} & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 4} \\ \mathbf{0}_{1\times 3} & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 3} \\ \mathbf{0}_{1\times 4} & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 2} \\ \mathbf{0}_{1\times 5} & \mathbf{Q}_{2:} & \mathbf{0}_{1\times 1} \\ \mathbf{0}_{1\times 6} & \mathbf{Q}_{2:} & \end{bmatrix} \begin{bmatrix} 1 \\ \lambda^1 \\ \lambda^2 \\ \lambda^3 \\ \lambda^4 \\ \lambda^5 \\ \lambda^6 \\ \lambda^7 \\ \lambda^8 \\ \lambda^9 \\ \lambda^{10} \\ \lambda^{11} \end{bmatrix} = \mathbf{S}(\boldsymbol{y})\boldsymbol{\lambda}. \tag{2.7}$$

Carefully note that the tail of $\mathbf{Q}_{1:}$ and $\mathbf{Q}_{2:}$ contain a linear term in $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ respectively. Therefore if the matrix has a non-trivial kernel, i.e. its determinant vanishes as per (2.2.3), both $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ satisfy the polynomial equations. This motivates the definition of the zero level set, $s(\boldsymbol{y}) = \det \mathbf{S}(\boldsymbol{y})$. This can be expressed as a function of $\boldsymbol{x}$ as,

$$(s \circ \boldsymbol{y})(\boldsymbol{x}) = \det \mathbf{S}(\mathbf{H}\boldsymbol{x}).$$

The method assumes the matrix $\mathbf{S}$ does not have a trivially vanishing determinant such as if the leading coefficients of both polynomials $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ are equal. In that case, extensions to the method exist that allow for the factoring of these terms [10]. Now we can define $\boldsymbol{\xi}_1$, the distance from the robot to the path, as the level set function $\boldsymbol{\xi}_1 := (s \circ \boldsymbol{y})(\boldsymbol{x})$. Observe that $\boldsymbol{\xi}_1$ is only dependent on $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Define $\boldsymbol{\xi}_2 := \dot{\boldsymbol{\xi}}_1 = \mathrm{d}_{\boldsymbol{y}}s \, \mathbf{H}\boldsymbol{f}(\boldsymbol{x})$.

To complete the control system, we must differentiate $\boldsymbol{\xi}_2$ in time. We expect (2.4) to emerge.

$$
\begin{aligned}
\dot{\boldsymbol{\xi}}_2 &= \frac{d}{dt}\left[\mathrm{d}_{\boldsymbol{y}}s\right]\mathbf{H}\boldsymbol{f}(\boldsymbol{x}) + \mathrm{d}_{\boldsymbol{y}}s\frac{d}{dt}\left[\mathbf{H}\boldsymbol{f}(\boldsymbol{x})\right]\\
&= (\mathbf{H}\boldsymbol{f}(\boldsymbol{x}))^\intercal\frac{d}{dt}[\mathrm{d}_{\boldsymbol{y}}s]^\intercal + \mathrm{d}_{\boldsymbol{y}}s\frac{d}{dt}\left[\mathbf{H}\boldsymbol{f}(\boldsymbol{x})\right]\\
&= (\mathbf{H}\boldsymbol{f}(\boldsymbol{x}))^\intercal\left[\mathrm{d}_{\boldsymbol{y}}^2 s\right]\mathbf{H}\boldsymbol{f}(\boldsymbol{x}) + (\mathrm{d}_{\boldsymbol{y}}s)\mathbf{H}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\boldsymbol{b}u\\
&= \mathcal{L}_{\boldsymbol{f}}^2\xi_1(\boldsymbol{x}) + \mathcal{L}_{\boldsymbol{b}}\mathcal{L}_{\boldsymbol{f}}\xi_1(\boldsymbol{x})\,u
\end{aligned}
$$

The controller implementation follows from Section 2.2.1.

2.2.3 Serret-Frenet Frame Approach

First we must impose an extra regularity condition, in addition to (2.2.1), in order for this controller to work.

Assumption 2.2.4 (Existence and Uniqueness of Nearest Point)**.** *Let* $C := \boldsymbol{\sigma}([0,1])$*. There exists an open, connected set* $\mathcal{U} \subset \mathcal{T}$ *such that,*

$$
(\forall y \in \mathcal{U}) : (\exists \lambda_* \in [0,1]) : \forall \lambda \in [0,1], \lambda \neq \lambda_* : \|y - \sigma(\lambda_*)\|_2 < \|y - \sigma(\lambda)\|_2\,.
$$

The path parameter λ_* is the nearest point on the curve to the robot. It is not difficult to see that this point also has the property that $\boldsymbol{\sigma}'(\lambda_*)$ is orthogonal to the error. A linear basis that spans $\mathbb{R}^2$ is built from λ_*. Let the first basis vector $\boldsymbol{e}_1 := \boldsymbol{\sigma}'(\lambda_*)/(\|\boldsymbol{\sigma}'(\lambda_*)\|_2)$. The second basis vector can be formed through a 90 degree rotation, such that $\boldsymbol{e}_2 := \boldsymbol{R}_{\frac{\pi}{2}}\boldsymbol{e}_1$. These basis vectors form the Serret-Frenet basis that spans $\mathbb{R}^2$. Relying on the intuition of previous derivations, we let $\boldsymbol{\xi}_1 := \boldsymbol{e}_2^\intercal(\boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{\sigma}(\lambda_*))$. In words $\boldsymbol{\xi}_1$ is the orthogonal tracking error with respect to the closest point on the curve. We complete the (partial) dynamical system state by differentiating $\boldsymbol{\xi}_1$ to find $\boldsymbol{\xi}_2$,

$$
\begin{aligned}
\boldsymbol{\xi}_2 &= \dot{\boldsymbol{\xi}}_1\\
&= \dot{\boldsymbol{e}}_2^\intercal(\boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{\sigma}(\lambda_*)) + \boldsymbol{e}_2^\intercal\left(\mathbf{H}(\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}u) - \boldsymbol{\sigma}'(\lambda_*)\dot{\lambda}\right)\\
&= \dot{\boldsymbol{e}}_2^\intercal(\boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{\sigma}(\lambda_*)) + \boldsymbol{e}_2^\intercal\left(\mathbf{H}(\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}u) - \boldsymbol{\sigma}'(\lambda_*)\dot{\lambda}\right).
\end{aligned}
$$

We require a result from differential geometry in order to continue.

Definition 2.2.5 (Curvature)**.** *The curvature of a regular,* C^2 *curve* $\sigma : \mathbb{R} \to \mathbb{R}^2$ *is*

$$
\kappa(\lambda) = \frac{\|\boldsymbol{\sigma}''(\lambda) \times \boldsymbol{\sigma}'(\lambda)\|_2}{\|\boldsymbol{\sigma}'(\lambda)\|_2^3} \tag{2.8}
$$

.

Our definition of the basis vectors as well as the restriction to planar curves (curves in $\mathbb{R}^2$) allows for a simplified definition of the curvature,

Lemma 2.2.6 (Signed Curvature of Planar Curve with Serret-Frenet Basis)**.** *The signed curvature is*

$$
\kappa(\lambda) = \frac{\boldsymbol{\sigma}''(\lambda)^\intercal \boldsymbol{e}_2}{\|\boldsymbol{\sigma}'(\lambda)\|_2^2}.
$$

Proof. Let θ be the angle between $\boldsymbol{\sigma}$ and $\boldsymbol{e}_1$.

$$\begin{aligned}
\kappa(\lambda) &= \frac{\left\|\boldsymbol{\sigma}''(\lambda) \times \boldsymbol{\sigma}'(\lambda)\right\|_2}{\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2^3} \\
&= \frac{\left\|\boldsymbol{\sigma}''(\lambda) \times \boldsymbol{e}_1\right\|_2}{\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2^2} \\
&= \frac{\left|\left\|\boldsymbol{\sigma}''(\lambda)\right\|_2 \sin\theta\right|}{\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2^2} \\
&= \frac{\left|\boldsymbol{\sigma}''(\lambda)^\mathsf{T}\boldsymbol{e}_2\right|}{\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2^2}.
\end{aligned}$$

$\square$

The curvature describes how much the curve's tangent changes. This is necessary to describe succinctly the derivative of the basis vectors. One more result will allow us to derive the derivatives of the basis vectors.

Lemma 2.2.7. *Let $\boldsymbol{v}_1, \boldsymbol{v}_2 : \mathbb{R} \to S^1$ such that both are C^2. Additionally, let $\boldsymbol{v}_2(\lambda) = \boldsymbol{R}_{\frac{\pi}{2}}\boldsymbol{v}_1(\lambda)$. Then, the derivatives of the vectors are related by a (parameter-varying) scalar multiple of the opposing vector such that*

$$\begin{aligned}
\boldsymbol{v}_1'(\lambda) &= \gamma(\lambda)\boldsymbol{v}_2 \\
\boldsymbol{v}_2'(\lambda) &= -\gamma(\lambda)\boldsymbol{v}_1.
\end{aligned}$$

Proof. Observe that $\forall \lambda \in \mathbb{R} : \boldsymbol{v}_1(\lambda)^\mathsf{T}\boldsymbol{v}_1(\lambda) \equiv 1$. Differentiating gives $\boldsymbol{v}_1'^\mathsf{T}\boldsymbol{v}_1 = 0$. Since $\{\boldsymbol{v}_1, \boldsymbol{v}_2\}$ spans $\mathbb{R}^2$, $\boldsymbol{v}_1'^\mathsf{T} = \gamma(\lambda)\boldsymbol{v}_2$. It is worth mentioning that $\gamma \in C^1$. The derivative $\boldsymbol{v}_2'$ follows similarly with the substitution $\boldsymbol{v}_2(\lambda) = \boldsymbol{R}_{\frac{\pi}{2}}\boldsymbol{v}_1(\lambda)$. $\square$

Corollary 2.2.8 (Derivative of Serret-Frenet Frame)**.** *For $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ as defined for the Serret-Frenet Controller,*

$$\begin{aligned}
\boldsymbol{e}_1'(\lambda) &= \kappa(\lambda)\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2 \boldsymbol{e}_2 \\
\boldsymbol{e}_2'(\lambda) &= -\kappa(\lambda)\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2 \boldsymbol{e}_1
\end{aligned}$$

.

Proof. Differentiate $\boldsymbol{e}_1$ with respect to λ explicitly,

$$\begin{aligned}
\frac{d\boldsymbol{e}_1}{d\lambda} &= \frac{d}{d\lambda}\left[\frac{\boldsymbol{\sigma}'(\lambda)}{\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2}\right] \\
&= \frac{1}{\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2^2}\left[\boldsymbol{\sigma}''(\lambda)\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2 - \boldsymbol{\sigma}'(\lambda)(\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2)^{-1}\boldsymbol{\sigma}''(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda)\right] \\
&= \frac{1}{\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2}\left[\boldsymbol{\sigma}''(\lambda) - (\boldsymbol{\sigma}''(\lambda)^\mathsf{T}\boldsymbol{e}_1)\boldsymbol{e}_1\right] \\
&= \frac{1}{\left\|\boldsymbol{\sigma}'(\lambda)\right\|_2}\left[\boldsymbol{\sigma}''(\lambda)^\mathsf{T}\boldsymbol{e}_2\right]\boldsymbol{e}_2
\end{aligned}$$

Likening the result with (2.2.7) and (2.2.6), we see the statement is proven. $\square$

A generalization of (2.2.7) exists in higher dimensions and can be found in any modern differential geometry text [6]. (2.2.8) permits the elimination of the first and last terms in $\boldsymbol{\xi}_2$. In practice, it may not always be possible to use — or even find — the closest point λ_* and therefore there may be some error introduced in the

system dynamics. An analysis of this effect is performed later. For now, we discard this term and observe that the vector $\boldsymbol{b}$ is in the kernel of $\boldsymbol{H}$ yielding a simple expression for $\boldsymbol{\xi}_2$,

$$\boldsymbol{\xi}_2 = \boldsymbol{e_2}^{\mathsf{T}}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})$$

This definition may be intuitively described as the projection of the output on the orthogonal vector. Define a new state vector $\boldsymbol{\xi}$ as before. Once again we would like to take the time derivative of $\boldsymbol{\xi}_2$ to complete the dynamical system. However two more Lemmas are required to help us take the time derivative of the basis vectors.

Lemma 2.2.9. *There exists a function $\varpi : \mathcal{U} \to [0,1]$ that generates the operating point λ_* from the position of the robot (the observation). The derivative of ϖ along the solutions of the system,*

$$\mathrm{d}_{\boldsymbol{y}}\varpi(\boldsymbol{y}) = \frac{\boldsymbol{e_1}^{\mathsf{T}}}{\|\boldsymbol{\sigma}'(\lambda_*)\|_2}.$$

Proof. Refer to Gill [3].   □

Computing the derivative of $\boldsymbol{\xi}_2$ we find,

$$\begin{aligned}
\dot{\boldsymbol{\xi}}_2 &= \dot{\boldsymbol{e}}_2^{\mathsf{T}}\mathbf{H}\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{e_2}^{\mathsf{T}}\mathbf{H}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\left[\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{b}u\right] \\
&= (\mathbf{H}\boldsymbol{f}(\boldsymbol{x}))^{\mathsf{T}}\dot{\boldsymbol{e}}_2 + \boldsymbol{e_2}^{\mathsf{T}}\mathbf{H}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\boldsymbol{b}u \\
&= (\mathbf{H}\boldsymbol{f}(\boldsymbol{x}))^{\mathsf{T}}\boldsymbol{e}_2'\left[(\mathrm{d}_{\boldsymbol{y}}\varpi)\mathbf{H}\boldsymbol{f}(\boldsymbol{x})\right] + \boldsymbol{e_2}^{\mathsf{T}}\mathbf{H}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\boldsymbol{b}u \\
&= (\mathbf{H}\boldsymbol{f}(\boldsymbol{x}))^{\mathsf{T}}\left[-\kappa(\lambda_*)\left(\|\boldsymbol{\sigma}'(\lambda_*)\|_2\right)\right]\boldsymbol{e}_1\left[(\mathrm{d}_{\boldsymbol{y}}\varpi)\mathbf{H}\boldsymbol{f}(\boldsymbol{x})\right] + \boldsymbol{e_2}^{\mathsf{T}}\mathbf{H}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\boldsymbol{b}u \\
&= -(\mathbf{H}\boldsymbol{f}(\boldsymbol{x}))^{\mathsf{T}}\boldsymbol{e}_1\frac{\boldsymbol{\sigma}''(\lambda)^{\mathsf{T}}\boldsymbol{e}_2}{\|\boldsymbol{\sigma}'(\lambda)\|_2}\frac{\boldsymbol{e_1}^{\mathsf{T}}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})}{\|\boldsymbol{\sigma}'(\lambda_*)\|_2} + \boldsymbol{e_2}^{\mathsf{T}}\mathbf{H}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\boldsymbol{b}u \\
&= -\left[\boldsymbol{e_1}^{\mathsf{T}}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})\right]^2\frac{\boldsymbol{e_2}^{\mathsf{T}}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^{\mathsf{T}}\boldsymbol{\sigma}'(\lambda_*)} + \boldsymbol{e_2}^{\mathsf{T}}\mathbf{H}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\boldsymbol{b}u \\
&= \mathcal{L}_{\boldsymbol{f}}^2\boldsymbol{\xi}_1(\boldsymbol{x}) + \mathcal{L}_{\boldsymbol{b}}\mathcal{L}_{\boldsymbol{f}}\boldsymbol{\xi}_1(\boldsymbol{x})u.
\end{aligned}$$

This result completes the partial dynamical system and allows for the construction of the linear controller. However, we would like to simplify the above system to be in terms of only $\boldsymbol{\xi}$, if possible. A useful result that combines the knowledge of our model and formulation of the basis vectors is required.

Lemma 2.2.10. *The translational velocity of the robot may be decomposed through projections on the basis vectors,*

$$v = \sqrt{\left(\boldsymbol{e_2}^{\mathsf{T}}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})\right)^2 + \left(\boldsymbol{e_1}^{\mathsf{T}}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})\right)^2}$$

which implies that,

$$(\boldsymbol{e_1}^{\mathsf{T}}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})) = \sqrt{v^2 - \boldsymbol{\xi}_2^2}.$$

Proof. The result follows from the fact that $\{\boldsymbol{e}_1, \boldsymbol{e}_2\}$ form an orthonormal basis and that $v = \|\mathbf{H}\boldsymbol{f}(\boldsymbol{x})\|_2$.   □

$$\dot{\boldsymbol{\xi}}_2 = -[\boldsymbol{e}_1^\mathsf{T}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})]^2 \frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)} + \boldsymbol{e}_2^\mathsf{T}\mathbf{H}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\boldsymbol{g}u$$

$$= -\left(v^2 - \boldsymbol{\xi}_2^2\right)\frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)} + \boldsymbol{e}_2^\mathsf{T}\mathbf{H}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\boldsymbol{g}u$$

$$= -\left(v^2 - \boldsymbol{\xi}_2^2\right)\frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)} + \boldsymbol{e}_2^\mathsf{T}\begin{bmatrix} -v\,\sin x_3 \\ v\,\cos x_3 \end{bmatrix}u$$

$$= -\left(v^2 - \boldsymbol{\xi}_2^2\right)\frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)} + \boldsymbol{e}_2^\mathsf{T}\boldsymbol{R}_{\frac{\pi}{2}}\begin{bmatrix} v\,\cos x_3 \\ v\,\sin x_3 \end{bmatrix}u$$

$$= -\left(v^2 - \boldsymbol{\xi}_2^2\right)\frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)} + \boldsymbol{e}_2^\mathsf{T}\boldsymbol{R}_{\frac{\pi}{2}}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})u$$

$$= -\left(v^2 - \boldsymbol{\xi}_2^2\right)\frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)} + \left(\boldsymbol{R}_{-\frac{\pi}{2}}\boldsymbol{e}_2\right)^\mathsf{T}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})u$$

$$= -\left(v^2 - \boldsymbol{\xi}_2^2\right)\frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)} + \boldsymbol{e}_1^\mathsf{T}\mathbf{H}\boldsymbol{f}(\boldsymbol{x})u$$

$$= -\left(v^2 - \boldsymbol{\xi}_2^2\right)\frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)} + u\sqrt{v^2 - \boldsymbol{\xi}_2^2}.$$

We now construct a general control signal of the form

$$u = \frac{u_{aux} + \left(v^2 - \boldsymbol{\xi}_2^2\right)\frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)}}{\sqrt{v^2 - \boldsymbol{\xi}_2^2}}$$

Notice that $|\boldsymbol{\xi}_2|$ must never equal v. Intuitively this means that the robot's speed must not be completely represented in the orthogonal direction — towards the path. This concern motivates an analysis on whether stability may be guaranteed while avoiding singularities in the control law at $|\boldsymbol{\xi}_2| = v$. Like in previous sections, we once again choose $u_{aux} = \boldsymbol{k}^\mathsf{T}\boldsymbol{\xi}$ and observe that this forms a closed loop linear system of the exact same form as (2.6). Again, the gains $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$ need to be negative to ensure stability of the closed loop dynamics. However, this result doesn't guarantee the system will not encounter the singularity described above. In summary, the robot's partial dynamics for the Serret-Frenet controller is,

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} \boldsymbol{\xi}_2 \\ -\left(v^2 - \boldsymbol{\xi}_2^2\right)\frac{\boldsymbol{e}_2^\mathsf{T}\boldsymbol{\sigma}''(\lambda)}{\boldsymbol{\sigma}'(\lambda)^\mathsf{T}\boldsymbol{\sigma}'(\lambda_*)} \end{bmatrix} + \begin{bmatrix} 0 \\ \sqrt{v^2 - \boldsymbol{\xi}_2^2} \end{bmatrix}u$$

Stability Analysis and Control Design

The final control system, with proportional linear feedback, is

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} 0 & 1 \\ \boldsymbol{k}_1 & \boldsymbol{k}_2 \end{bmatrix}\boldsymbol{\xi}. \tag{2.9}$$

We begin with a necessary result from Dynamical System Theory.

Theorem 2.2.11 (Positive Invariance). *Let $\Omega \subset \mathbb{R}^2$ be an open, connected set and $V : \Omega \to \mathbb{R}$ be continuously differentiable such that*

$$\dot{V}(\boldsymbol{\xi}) \leq 0, \boldsymbol{\xi} \in \Omega.$$

Then Ω is made positively invariant by $V(\boldsymbol{\xi})$. That is all orbits that begin in Ω stay in Ω.

Now we prove the main result,

Theorem 2.2.12 (Singularity-Free Orbits for Serret Frenet Control)**.** *For any $M > 0$, there exists a choice of gains — $\boldsymbol{k}_1$, $\boldsymbol{k}_2$ — such that an open, connected set Ω is made positively invariant by some Lyapunov function $V(\boldsymbol{\xi})$ under the closed loop dynamics of* (2.9). *Furthermore, the set Ω has the properties*

1. $\sup_{\boldsymbol{\xi} \in \Omega} |\boldsymbol{\xi}_1| = M$

2. $\sup_{\boldsymbol{\xi} \in \Omega} |\boldsymbol{\xi}_2| = v$.

Proof. We use the method of undetermined coefficients over a quartic positive definite form. Let $V(\boldsymbol{\xi}) = \boldsymbol{\xi}_1^4 + a\boldsymbol{\xi}_1^2\boldsymbol{\xi}_2^2 + b\boldsymbol{\xi}_2^4$ where $a, b \in \mathbb{R}$ and are positive. We construct Ω as the open, connected, sub-level set of V so that $\Omega = \{\boldsymbol{\xi} : V(\boldsymbol{\xi}) < D\}$. We would like to impose restrictions on a, b, c, and D so that Ω is made positively invariant by V and so that Ω has the desired properties. Impose $b = Dv^{-4}$ to ensure that $|\boldsymbol{\xi}_2| < v$. We attempt to find further conditions so that $\dot{V} < 0$,

$$
\begin{aligned}
\dot{V} &= 4\boldsymbol{\xi}_1^3\dot{\boldsymbol{\xi}}_1 + 2a\left[\boldsymbol{\xi}_1\dot{\boldsymbol{\xi}}_1\boldsymbol{\xi}_2^2 + \boldsymbol{\xi}_1^2\boldsymbol{\xi}_2\dot{\boldsymbol{\xi}}_2\right] + 4b\boldsymbol{\xi}_2^3\dot{\boldsymbol{\xi}}_2 \\
&= 4\boldsymbol{\xi}_1^3\boldsymbol{\xi}_2 + 2a\left[\boldsymbol{\xi}_1\boldsymbol{\xi}_2^3 + \boldsymbol{k}_1\boldsymbol{\xi}_1^3\boldsymbol{\xi}_2 + \boldsymbol{k}_2\boldsymbol{\xi}_1^2\boldsymbol{\xi}_2^2\right] + 4b\boldsymbol{k}_1\boldsymbol{\xi}_1\boldsymbol{\xi}_2^3 + 4b\boldsymbol{k}_2\boldsymbol{\xi}_2^4 \\
&= (4 + 2a\boldsymbol{k}_1)\boldsymbol{\xi}_1^3\boldsymbol{\xi}_2 + 2\left(a + 2b\boldsymbol{k}_1\right)\boldsymbol{\xi}_1\boldsymbol{\xi}_2^3 + \left(2a\boldsymbol{\xi}_1^2\boldsymbol{\xi}_2^2 + 4b\boldsymbol{\xi}_2^4\right)\boldsymbol{k}_2 \\
&= (4 + 2a\boldsymbol{k}_1)\boldsymbol{\xi}_1^3\boldsymbol{\xi}_2 + 2\left(a + 2Dv^{-4}\boldsymbol{k}_1\right)\boldsymbol{\xi}_1\boldsymbol{\xi}_2^3 + \left(2a\boldsymbol{\xi}_1^2\boldsymbol{\xi}_2^2 + 4Dv^{-4}\boldsymbol{\xi}_2^4\right)\boldsymbol{k}_2.
\end{aligned}
$$

Impose the requirement that cross-terms vanish identically. This gives the restrictions on a and $\boldsymbol{k}_1$,

$$
\boldsymbol{k}_1 = -\frac{v^2}{\sqrt{D}}
$$

$$
a = 2\frac{\sqrt{D}}{v^2}.
$$

Then $V(\boldsymbol{\xi}) = \boldsymbol{\xi}_1^4 + 2\frac{\sqrt{D}}{v^2}\boldsymbol{\xi}_1^2\boldsymbol{\xi}_2^2 + \frac{D}{v^4}\boldsymbol{\xi}_2^4$. Interestingly, this is the square of a quadratic form. Specifically,

$$
V(\boldsymbol{\xi}) = \left[\boldsymbol{\xi}_1^2 + \left(\frac{\sqrt{D}}{v^2}\right)\boldsymbol{\xi}_2^2\right]^2.
$$

We now know that $\dot{V} = \left(2a\boldsymbol{\xi}_1^2\boldsymbol{\xi}_2^2 + 4Dv^{-4}\boldsymbol{\xi}_2^4\right)\boldsymbol{k}_2$. Now, $\dot{V} < 0$ if $\boldsymbol{k}_2 < 0$. We therefore impose $\boldsymbol{k}_2 < 0$. Ω is now made positively invariant as a consequence of (2.2.11). Specify $D = M^4$ and find

$$
\sup_{\boldsymbol{\xi} \in \Omega} |\boldsymbol{\xi}_1| = M
$$

$$
\sup_{\boldsymbol{\xi} \in \Omega} |\boldsymbol{\xi}_2| = v
$$

as required. Substituting the restricted a, b, c, and D back into our equation we find in summary

$$
\boldsymbol{k}_1 = -\left(\frac{v}{M}\right)^2
$$

$$
\boldsymbol{k}_2 < 0
$$

$$
\Omega = \left\{\boldsymbol{\xi} : \boldsymbol{\xi}_1^2 + \left(\frac{M}{v}\right)^2\boldsymbol{\xi}_2^2 < M^2\right\}
$$

$$
V(\boldsymbol{\xi}) = \left[\boldsymbol{\xi}_1^2 + \left(\frac{M}{v}\right)^2\boldsymbol{\xi}_2^2\right]^2.
$$

$\square$

Corollary 2.2.13. *For any $M > 0$, an open, connected set of solutions that have initial conditions that observe*

$$\boldsymbol{\xi}(0)^{\mathsf{T}} \begin{bmatrix} 1 & 0 \\ 0 & \left(\frac{M}{v}\right)^2 \end{bmatrix} \boldsymbol{\xi}(0) < M^2$$

will asymptotically converge to $\mathbf{0}$ without encountering the singularity $|\boldsymbol{\xi}_2| = v$.

Proof. Proof follows from the fact that the closed loop system in (2.9) is stable if and only if both gains are negative, which they are in (2.2.12). $\qquad\square$

Observe that (2.2.13) *does not* guarantee a singularity-free orbit for all initial conditions using a single choice of gains. It simply states that a certain set, that is chosen a-priori, is made positively invariant by a choice of gain. In fact, if one were to let $M \to \infty$ then the gains will tend to 0 and the dynamical system degenerates so that closed loop stability is not maintained. Also observe the simplicity of the set Ω that is constructed. It is easy to test in practice whether or not the system is within the singularity-free region for a particular controller design. This result motivates a control design methodology for the Serret-Frenet controller. By choosing a maximum allowable orthogonal error M, the gain $\boldsymbol{k}_1 = -(\frac{v}{M})^2$ is determined. Observe that (2.4) is a second order system with characteristic polynomial $s^2 - \boldsymbol{k}_2 s - \boldsymbol{k}_1$. Then the harmonic frequency and damping ratio are,

$$\omega = \sqrt{-\boldsymbol{k}_1}$$
$$\zeta = \frac{-\boldsymbol{k}_2}{2\sqrt{-\boldsymbol{k}_1}}.$$

(2.2.13) limits the choice of $\boldsymbol{k}_1$ so we can only determine the damping ratio. We would like, for simplicity, to choose to have no overshoot while still optimizing for the fastest convergence. This implies we need $\zeta = \sqrt{2}/2$. This design decision restricts $\boldsymbol{k}_2 = -\sqrt{-2\boldsymbol{k}_1}$. The implementation used in this paper chooses $M = 0.1$ since the controller expects a very little amount of error.

Chapter 3

Planning

Path planning involves the generation of waypoints and curves for the purpose of arriving at some goal region, $\mathcal{G}$. This paper only describes the Rapidly exploring Randomized Tree (RRT) algorithm. A majority of the investigation in planning occurred in the polynomial spline generation phase.

3.1 RRT

The Rapidly exploring Randomized Tree (RRT) algorithm is made up of the following steps [5],

1. Initialize tree $R = \{\boldsymbol{x}_0\}$ with the initial point.

2. Randomly sample a candidate point $\boldsymbol{x} \in \mathcal{T}$.

3. Find the nearest point $\boldsymbol{x}_n \in \mathcal{T}$ to x, based on some metric over $\mathcal{T}$.

4. Add an edge $(\boldsymbol{x}_n, \boldsymbol{x})$ if the path between them is collision free.

5. If $\boldsymbol{x} \in \mathcal{G}$ halt, otherwise repeat from (2).

The tree grows in a random fashion from the initial point, rapidly expanding in $\mathcal{T}$ until the goal can be reached. Once the algorithm completes, the programmer need only back-trace from the goal point towards the initial point; trees have a unique path from any node to the root. There are a few design decisions that can be made here in order to improve performance of the algorithm. One is to store obstacles in what is known as an R-Tree. R-Trees — a tree whose name stems from rectangle, not rapidly nor randomized — are a data structure that stores data according to their position in space. R-Tree creates bins that store objects with bounding rectangles that are near one another. This data structure has an *average* time complexity that is $\mathcal{O}(\log_2 N)$ where N is the number of elements — in our case, obstacles — for searching for an element in the two dimensional case. As a result, it scales well even for large spatial domains where many obstacles may exist.

One reason this algorithm was developed was to help developers incorporate constraints on the generated path. Recall that the effectiveness of the transverse feedback linearization is related to the curvature of the path. The higher the curvature, the more sensitive the controller is to small errors. Moreover, it may be the case that (2.2.4) fails to hold. One way to easily restrict the curvature is to ensure the angle of edge (x_n, x) doesn't deviate strongly from the angle of the line (x_m, x_n). That is, the angle between any two successive edges is restricted to $\pi \pm \theta_{lim}$.

Another approach attempted is to enforce a distance constraint that is related to the angle. Unfortunately, this does not work as well in practice, as it greatly reduces the exploration performance of RRT [1]. RRT also does not by itself provide guarantees about the 'directness' of the path generated. This can changed by creating a bias towards sampling near $\mathcal{G}$[7][1]. We chose to randomly sample candidate points from a truncated-normal distribution centred about $\mathcal{G}$. The variance was chosen to ensure about 70-95% of points are within the task space. This allowed for biased sampling that still properly explored the domain.

3.2 Polynomial Splining

Polynomial splining is a technique by which possibly distinct polynomials are stitched together to preserve a certain level of continuity at the stitch-points. Consider the polynomial $\boldsymbol{\sigma}(\lambda) = \sum_{k=0}^{N} (a_k, b_k)^{\mathsf{T}} \lambda^k$. Normally we would like this polynomial to evaluate to the start and end points at $\lambda = 0$ and $\lambda = 1$ respectively. This clearly enforces a linear equality constraint. Continuity conditions are imposed on other polynomials that touch this polynomial, at the end points, by first enforcing that they evaluate to the same point at some other λ (possibly the same value). Then derivatives are taken, as many as the order of continuity is required, and evaluated at the endpoints and set equal to one other. This results in a homogeneous linear equation. This set of linear equations may be solved by any standard algorithm, such as LU decomposition.

In the case where there are fewer equations than unknowns, the system is under-determined. Although this can be solved using Gaussian elimination, it is preferred to use the Moore-Penrose pseudo-inverse (least squares) instead. This finds a solution that also minimizes the norm of the vector, in this case the vector of coefficients.

There is a problem with the above approach in the context of path planning. Since these polynomials are generated from waypoints generated by RRT, we know that the only guarantees we have about the waypoints are that the edges between them are collision free. There is no reason to think that the polynomial may not deviate from this edge, which it normally does, in a manner that may collide with an object. Of course, the easy way to deal with this is to test manually whether the polynomial intersects the object through a linear approximation of it. This is slow. Instead, we design a new methodology that attempts to fit the polynomial to the edges, instead of restricting the polynomial to the waypoints.

3.2.1 Loosely Constrained Spline

Let $W = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_f\} \subset \mathcal{T}$ be the set of waypoints and let the polynomial be defined as before. We would like to construct a polynomial that fits every point W within an error δ while minimizing the deviation between it and the edge between waypoints. A simple way of performing this is to parameterize the edges with the same parameter as the polynomial, and construct a least squares matrix for these intermediate points. Conditions may be constructed in the form of $\boldsymbol{\sigma}_1(\lambda) - \boldsymbol{w}_{(1,i)} < \delta$ and $w_{(1,i)} - \boldsymbol{\sigma}_1(\lambda) < \delta$. The same condition is also used to restrict the second component of $\boldsymbol{\sigma}$. Note that this form enforces $\|\boldsymbol{\sigma}(\lambda) - \boldsymbol{w}_i\|_\infty < \delta$. With linear constraints, we can now formulate the splining problem as a linear-inequality constrained least squares problem. This can be solved in MATLAB using `lsqlin`; any optimization algorithm that allows for constraints may perform well in this area.

By permitting this freedom, the curve closely follows the edges of RRT and is unlikely to introduce collisions in the generated path. Of course, the path does deviate from the waypoint indicated a trade-off in this method. Further, the cost of the splining rises dramatically as a linear algebra problem is now converted into an optimization problem.

Chapter 4

Case Study

4.1 Baseline Study

An initial performance comparison on the controllers was performed in MATLAB with a quintic polynomial path generated using the technique described in Section 3.2.1. The polynomial has small coefficients with enough curvature to have interesting behaviour but not too high curvature that may adversely affect all controllers. The path can be seen in both Figure 4.2 and Figure 4.1. The tracking controller, as seen in Figure 4.1, seems to successfully track the path. However, the path is traced in less than one second! This is where the scaling parameter — T_f in Section 2.1 — comes into play. However, all this parameter can do is force the arrival time to be T_f. This isn't ideal. For the mock world study, T_f was estimated using knowledge of the distance between the start and goal points on the map.
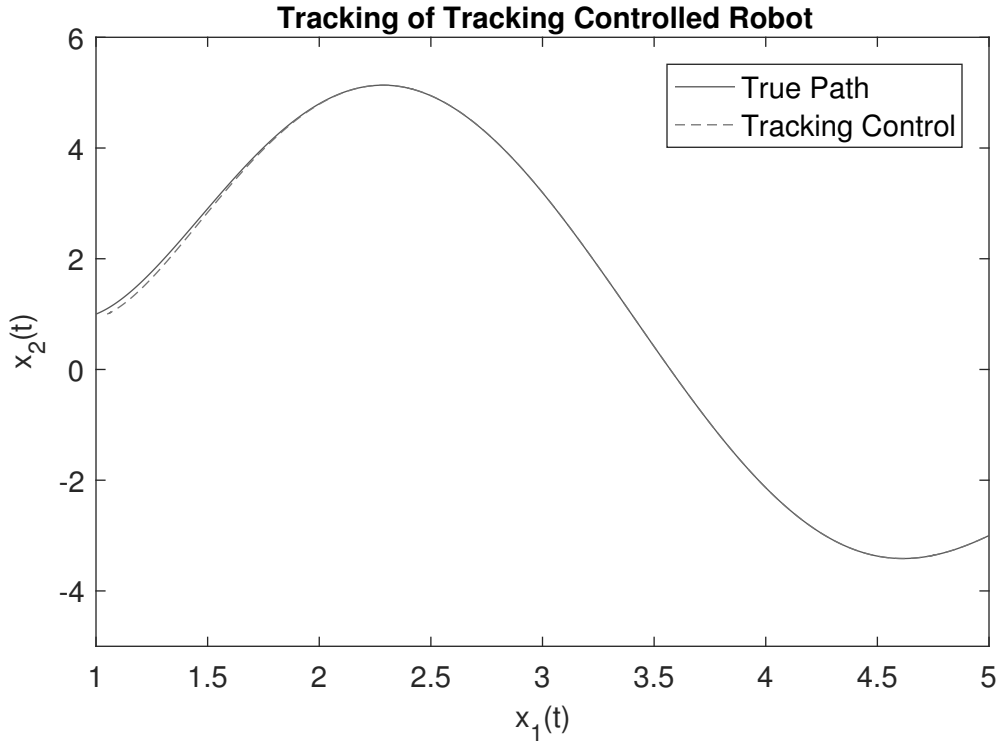
Figure 4.1: The track of the robot controlled by Tracking Control in task space. Driving towards and onto path.

The Serret-Frenet and Sylvester controllers should exhibit exponential convergence to the path in $\boldsymbol{\xi}$-space as the gains are chosen to satisfy the Routh-Hurwitz criterion applied to the characteristic polynomial of (2.6).

Gains are specifically chosen to ensure the Serret-Frenet controller doesn't exceed an orthogonal tracking error of 0.1, as described in Section 2.2.3. The Sylvester approach uses the exact same gains to provide a fair comparison. Figure 4.2 shows the resulting path of both robots in a situation where they are on the path but not facing in the right direction — the robots start with a heading error of over 45 degrees. Notice that both methodologies converge and remain on the path no matter the behaviour of the curve. Figure 4.2 isn't precise
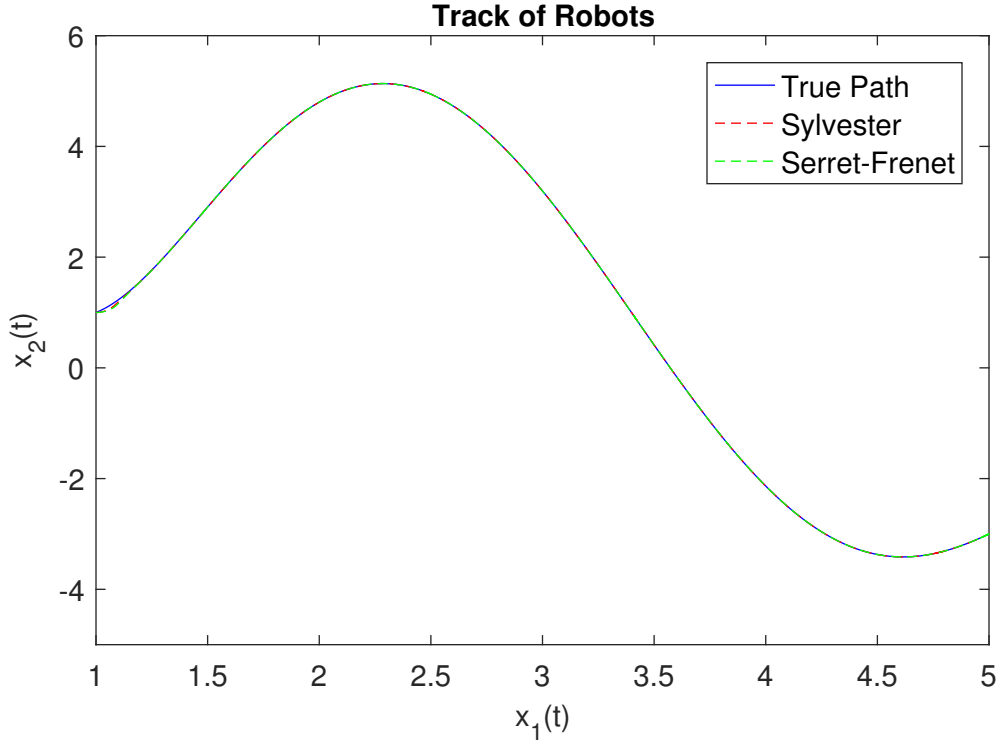
Figure 4.2: The track of the robots in task space driving towards and on the path.

enough to verify performance, so we turn to the orthogonal tracking error plot in Figure 4.3. It is now apparent how far away both methods go before converging to the path tangentially. Interestingly, the Sylvester controller outperforms the Serret-Frenet controller in both maximum deviation and rate of convergence. At face value, this indicates that the Sylvester controller is the one to choose.

Figure 4.4 indicates a possible problem. The Sylvester controller generates states that are orders of magnitude larger than the Serret-Frenet controller, which may mean that the Sylvester controller generates large control signals. Keep in mind that the error in position was less than 10^{-1}! However, the control signals generated are still comparable to the Serret-Frenet controller, as shown in Figure 4.5. This must mean that the Lie derivative(s) in the Sylvester method are of a similar magnitude of the state. However, there is no reason to expect this method will extend well for large polynomials over a large domain since the scaling of large magnitude states into relatively small control signals may lead to a non-negligible loss of precision. This is of serious concern for practical implementations of these controllers for high-order polynomials and/or large paths.

This isn't the main dimension where the Sylvester controller falls short. Table 4.1 collates the time-cost of executing the control-generation functions for both controllers, measured in MATLAB. The gradient descent algorithm is included so as to provide a useful baseline of performance. Notice that the gradient descent is the dominating cost in the Serret-Frenet controller. The Sylvester controller, on the other hand, performs poorly. In fact, data — not shown here — indicates that 62% of the time is spent just calculating the $\boldsymbol{\xi}$ state vector. Any implementation of the Sylvester controller can perform, at best, around 30 steps per second. The reader would like to take note that the machine used for benchmarking uses an Intel processor running at 2.50GHz. An embedded processor is unlikely to effectively execute the Sylvester controller. Unfortunately, these are
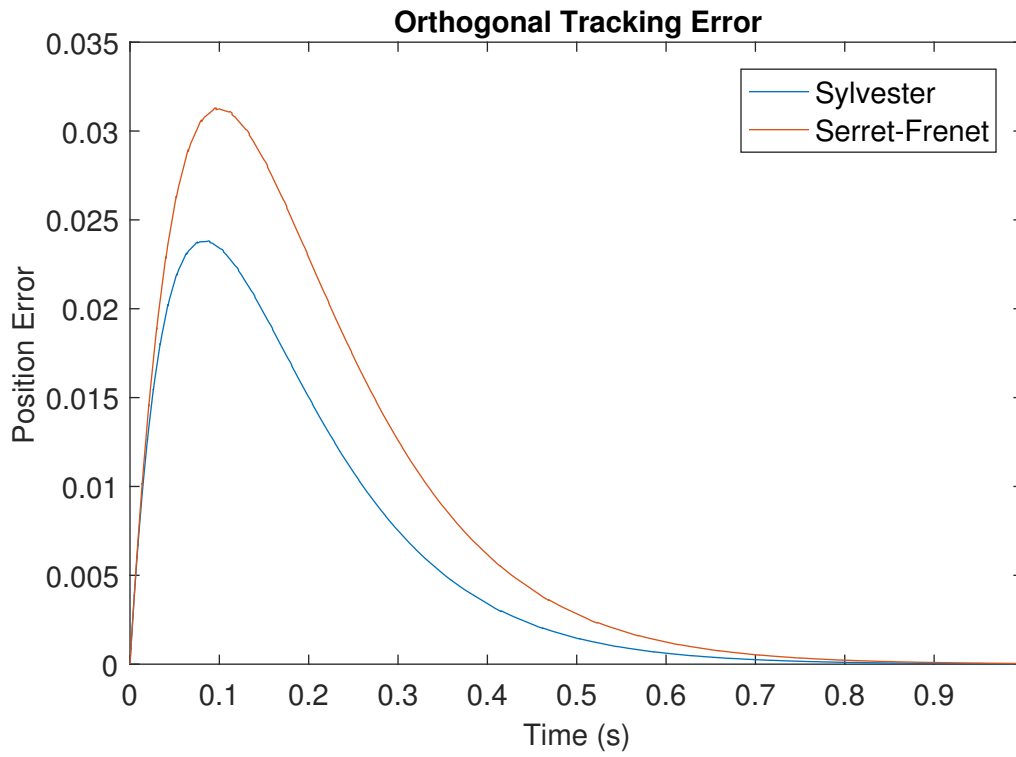
Figure 4.3: *Comparison of tracking error. Notice that both experience exponential convergence, however the Sylvester method seems to converge faster.*
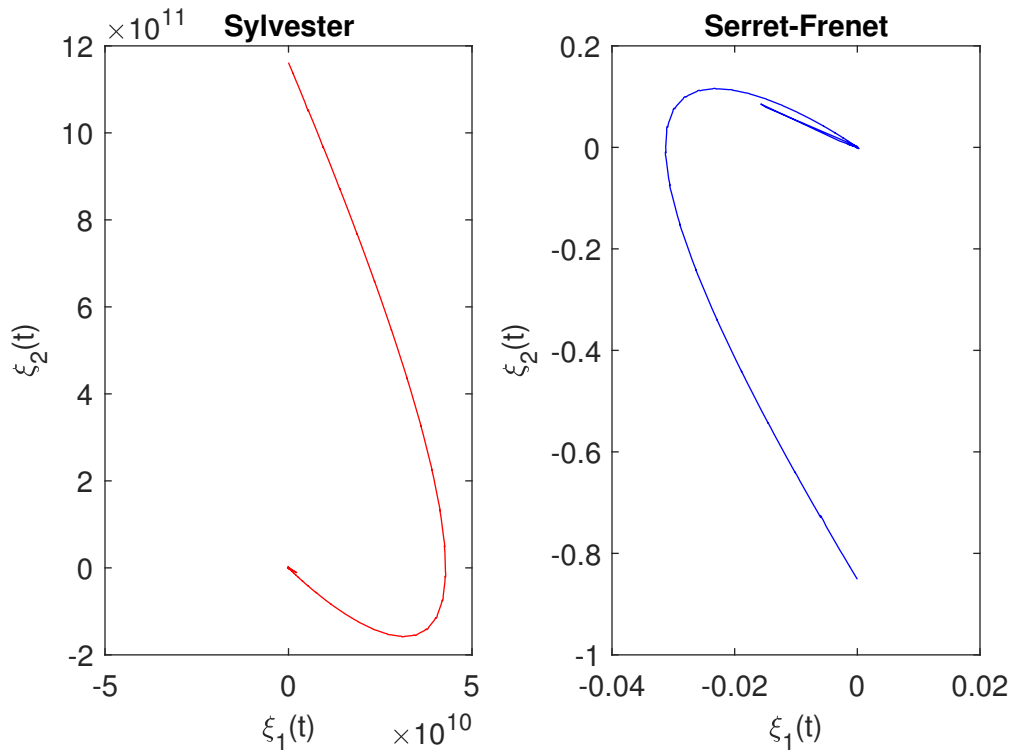

Figure 4.4: *The linearized state $\boldsymbol{\xi}$ for both controllers. Take careful note of the scale of the axes!*
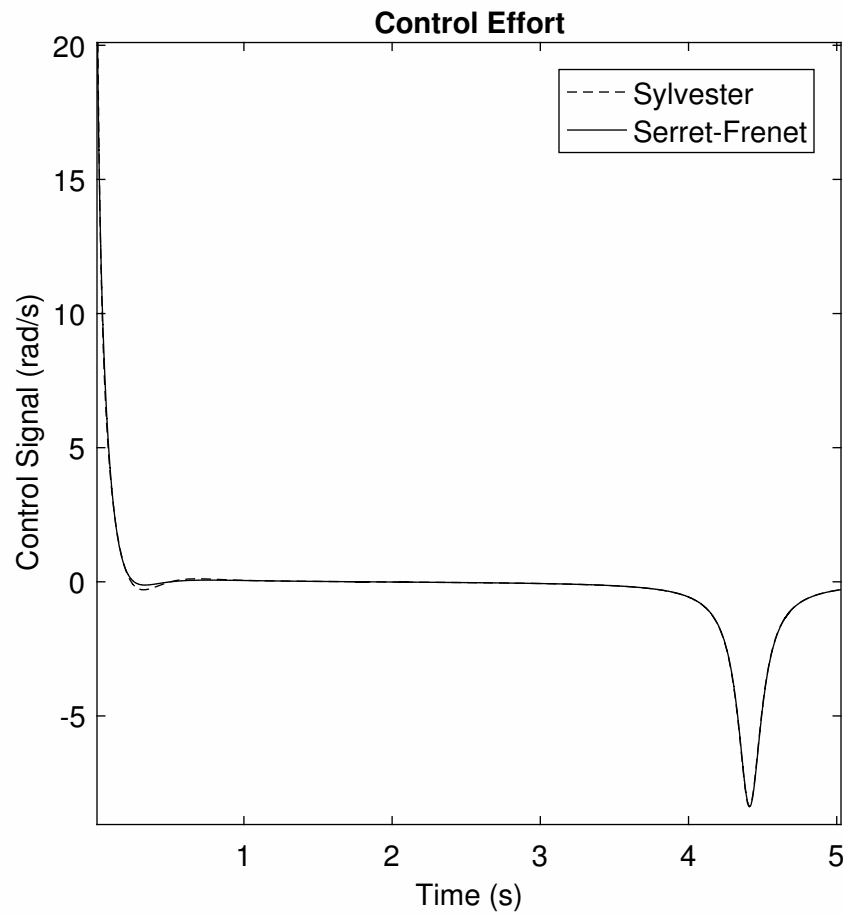
Figure 4.5: Control effort of the Sylvester and Serret-Frenet controllers. They are almost indistinguishable.

timing benchmarks and are, as a result, not easily transferable to other hardware. An analysis of floating point operations is required. Both controllers were implemented in C++ and compiled with GCC with optimization

Function of Interest	Total Time Used (s)	Number of Calls	Estimated Cost (ms)
Gradient Descent	11.339	2017	5.6217
Serret-Frenet	12.812	2017	6.3520
Sylvester	61.338	2095	29.278

Table 4.1: Time cost of the control generation functions. The gradient descent algorithm cost is included in their timing results.

turned off in order to correctly estimate number of floating point operations. Table 4.2 lists the results. The Serret-Frenet controller is expected to be 3 orders of magnitude faster just by a rough count of floating point operations. This is a rough estimate as it doesn't account for optimization.

Function of Interest	Floating Point Operation Count	Comments
Serret-Frenet	316	From evaluating polynomial and its derivatives.
Sylvester	111159	From the level set evaluation functions.

Table 4.2: Floating point operation counts. Manually counted by assembly inspection and regular expression searches, cross-verified with actual source code. Values may vary by single-digit percentiles but their order of magnitude should remain the same.

4.2   Mock World Study

In this section, the controllers are pitted against one another in a mock real world use case. The University of Waterloo campus map was mocked, treating buildings as rectangular obstacles, as shown in Figure 4.6. The robots begin at the bottom of campus and are told to make their way to the top of campus. An implementation of RRT is used to discover paths to the goal. The spline generated is a sequence of quintic polynomials stitched together preserving C^2 continuity as required. The path followed by both the Serret-Frenet controller and the Tracking controller are shown in Figures 4.7 and 4.8. The asterisks occur at points where the robot made the decision to change the path, i.e. initiate path planning.

Both controllers achieve, at face value, the tracking objective. We can only know for sure by verifying the tracking error. Figures 4.9 and 4.10 illustrate the tracking error. The Serret-Frenet controller maintains an almost constant error that is negligible, since the path is planned such that the robot is already correctly configured on the path. This corroborates our knowledge that the Serret-Frenet controller treats the path as invariant. The tracking controller also achieves relatively good tracking, except at the path planning event points. This may occur because the path plan begins at the robot's position, instead of the lead point. Barring this issue, the tracking controller performs well. However, notice that we can achieve just as good tracking without the velocity control introduced by the tracking controller. Moreover, the tracking controller requires a great deal of tuning.   The reader may be wondering why the Sylvester controller was not tested in this benchmark. There are a few reasons for this:

1. The control level set functions could not be optimized by the compiler (GCC).

2. Integration tolerances could not be achieved.

The components of $\boldsymbol{\xi}$ are at least 10 orders of magnitude larger than the Serret-Frenet controller (or, for that matter, any reasonable controller) as seen in Figure 4.4. The control signal however is still reasonable, shown in Figure 4.5, and in fact closely follows that of the Serret-Frenet controller. With that said, this controller is at risk of numerical errors due to relative magnitude of the lie derivatives/state and the resulting control signal. This may be why integration tolerances could not be achieved in the larger polynomial case.
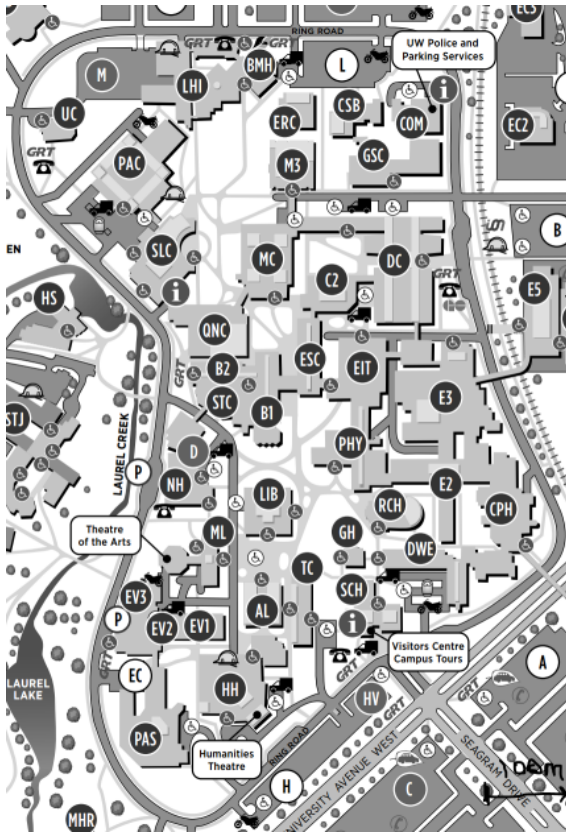
(a) Original campus map.
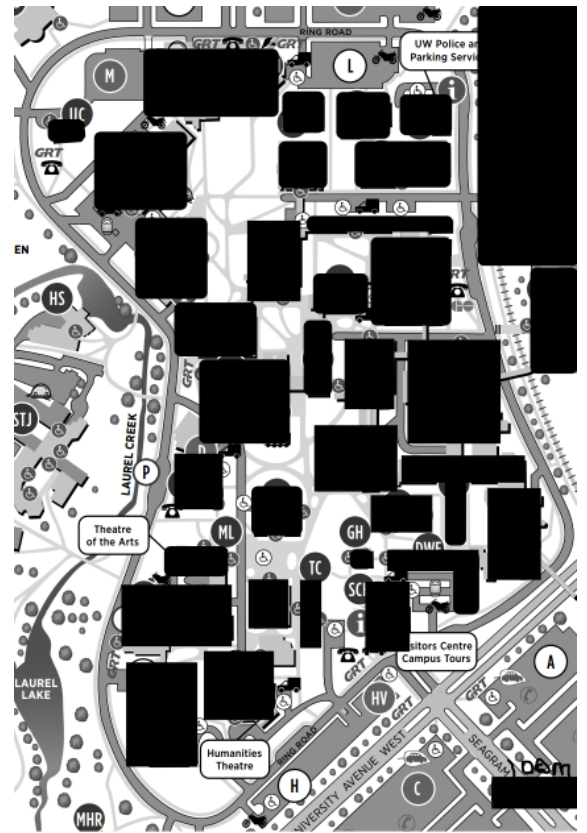
(b) Original campus map with rectangular obstacles.

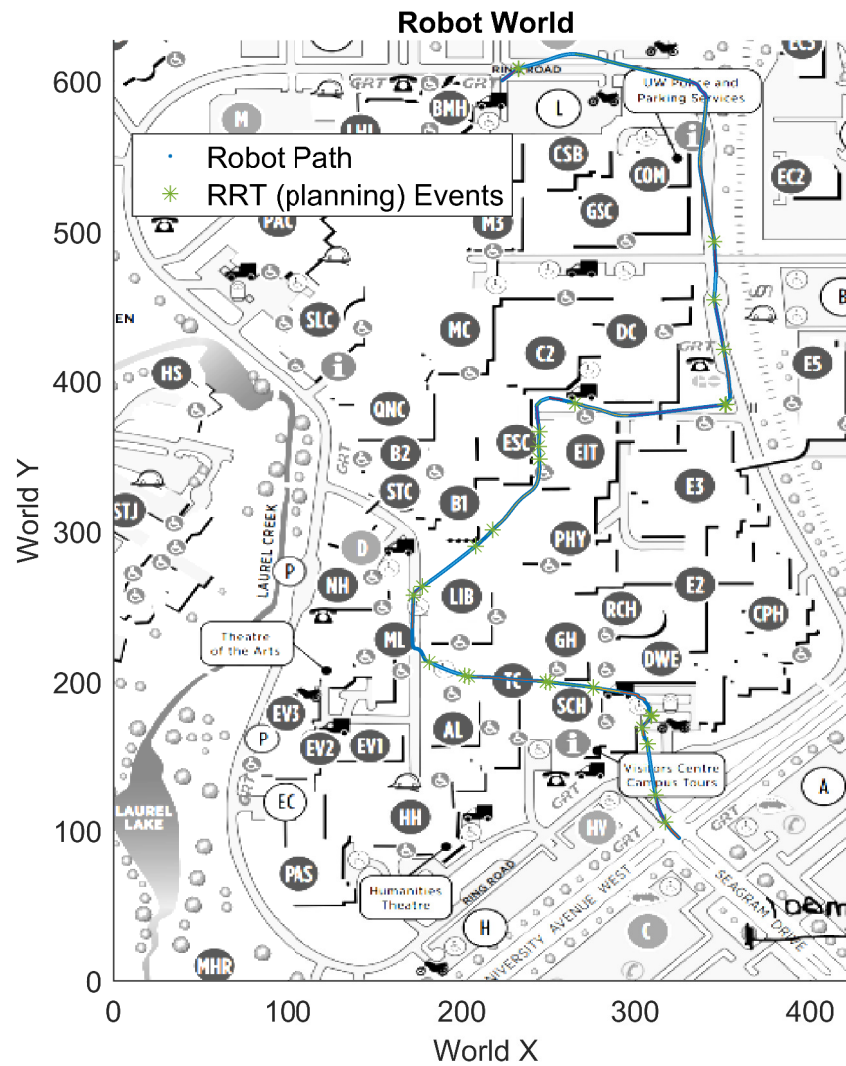Figure 4.6: Mock campus world model. 1 pixel equals approximately 0.64m.

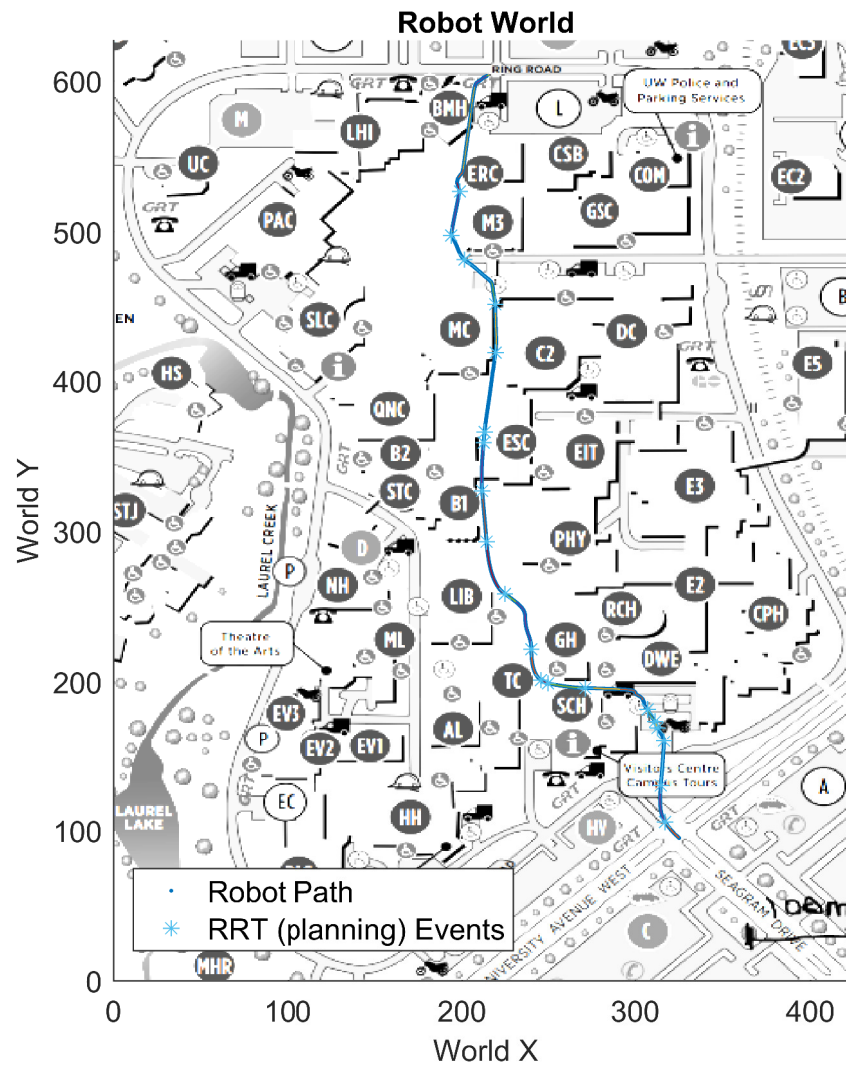Figure 4.7: Path followed by Serret-Frenet controller.
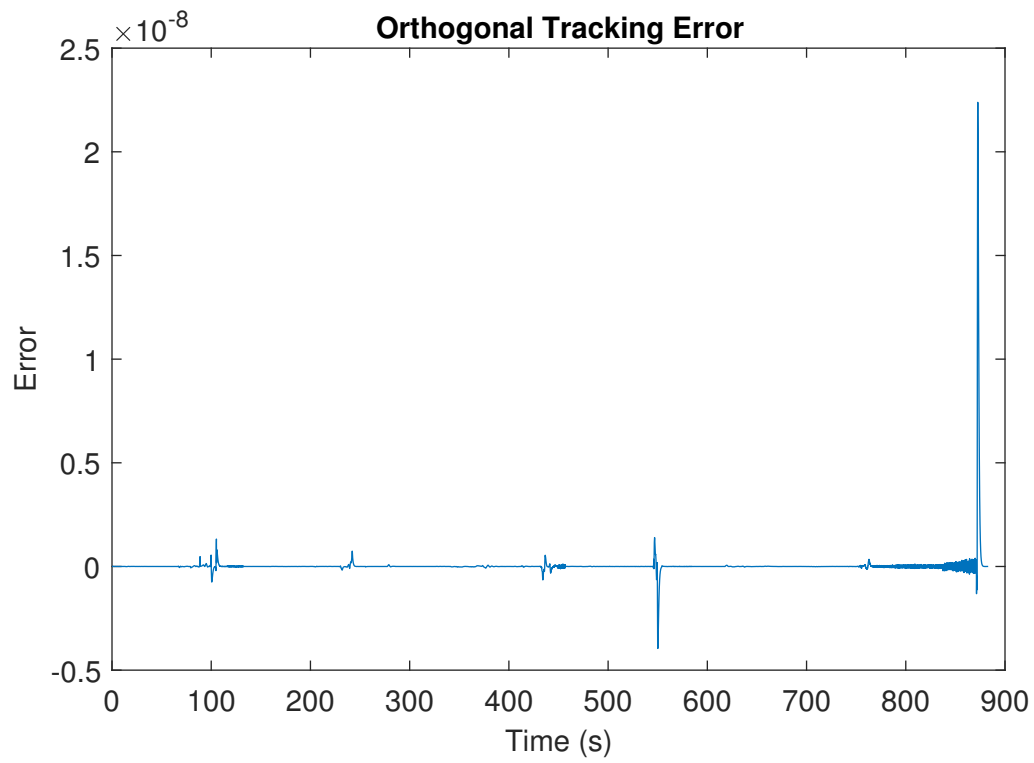
Figure 4.8: Path followed by Tracking controller.

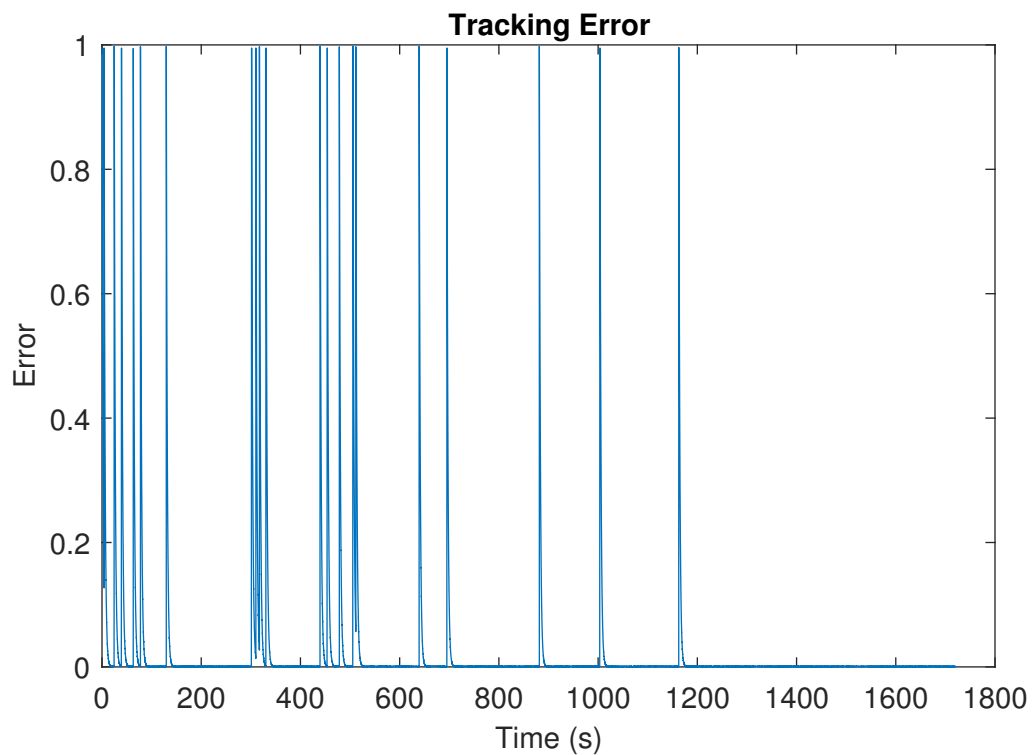*Figure 4.9: Error (orthogonal only) in Serret-Frenet controller.*

Figure 4.10: Error in Tracking controller.

4.3   Conclusions

A set of path following controllers were implemented in both MATLAB and C++ and verified in a relatively large scale mock example. A reformulation of the splining problem in terms of looser constraints was proposed. An analysis of the Serret-Frenet controller was performed, with the additional development a control design methodology that allows for a control engineer to design gains based on a-priori restrictions on the orthogonal tracking error.

(2.2.12) does provoke a few interesting lines of research. Firstly, it motivates the use of a dynamic gain, that changes as the system evolves deeper into the invariant set. This approach may be able to speed convergence towards the path while ensuring the robot doesn't veer outside into the unknown domain. In addition to this, the Theorem seems to indicate that a design methodology may exist that incorporates conditions on $\boldsymbol{\xi}_2$ as well! Restrictions on $\boldsymbol{\xi}_2$ are effectively equivalent to preventing the robot from approaching too steeply towards the path. This may have applicability in domains where the robot cannot (and maybe even should not) make a bee-line towards the path for safety reasons. A simple example is a car which must switch from the slow lane to the fast lane on a wide highway. Even on an open road, it is safer for both passengers and other drivers to approach in a manner that reduces the turn rate.

The Sylvester controller, though mathematically elegant, doesn't seem to be well suited for practical implementation. In fact, my compiler couldn't optimize any of the Lie derivative and Level set functions for the Sylvester controller. Instead, the compiler hung. This and the fact that the Sylvester controller is expensive in terms of floating point operations, indicates a lack of practicality. Even in simulation, the Sylvester controller generated unreasonably large linearized states that were then scaled down with possible loss of precision. It is apparent that the Serret-Frenet controller is superior.

References

[1] K. Yang et al. "Spline-Based RRT Path Planner for Non-Holonomic Robots". In: *J Intell Robot Syst* 73 (2014), pp. 763–782.

[2] *Boost C++ Libraries*. http://www.boost.org/.

[3] R.J. Gill, D. Kulic, and C. Nielsen. "Spline Path Following for Redundant Mechanical Systems". In: *IEEE Transactions on Robotics* 31 (2015), pp. 1378–1392.

[4] A. Hladio, C. Nielsen, and D. Wang. "Path following for a class of mechanical systems". In: *IEEE Trans. Control Syst. Technol.* 21.6 (2013), pp. 2380–2390.

[5] S. Karaman and E. Frazzoli. "Sampling-based Algorithms for Optimal Motion Planning". In: *The International Journal of Robotics Research* 30 (7 2011), pp. 846–894.

[6] W. Kuhnel. *Differential Geometry: Curves—Surfaces—Manifolds Second Edition*. American Mathematical Society, 2006.

[7] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[8] Phil Nash. *Catch Test Framework*. https://github.com/philsquared/Catch.

[9] C. Nielsen and M. Maggiore. "Output stabilization and maneuver regulation: A geometric approach". In: *System Control Letters* 55 (2006), pp. 418–427.

[10] T.W. Sederberg, D.C. Anderson, and R.N. Goldman. "Implicit Representation of Parametric Curves and Surfaces". In: *Computer Vision, Graphics and Image Processing* 28 (1984), pp. 72–84.